In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, roc_auc_score, precision_recall
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import tensorflow as tf

pd.options.display.max_columns = None
```

In [ ]:
```
df_covid = pd.read_csv('./Covid_clean.csv')
```

Contenido del Dataset

The dataset was provided by the Mexican government (link). This dataset contains an enormous number of anonymized patient-related information including pre-conditions. The raw dataset consists of 21 unique features and 1,048,576 unique patients. In the Boolean features, 1 means "yes" and 2 means "no". values as 97 and 99 are missing data.

- sex: 1 for female and 2 for male.
- age: of the patient.
- classification: covid test findings. Values 1-3 mean that the patient was diagnosed with covid in different
- degrees. 4 or higher means that the patient is not a carrier of covid or that the test is inconclusive.
- patient type: type of care the patient received in the unit. 1 for returned home and 2 for hospitalization.
- pneumonia: whether the patient already have air sacs inflammation or not.
- pregnancy: whether the patient is pregnant or not.
- diabetes: whether the patient has diabetes or not.
- copd: Indicates whether the patient has Chronic obstructive pulmonary disease or not.
- asthma: whether the patient has asthma or not.
- inmsupr: whether the patient is immunosuppressed or not.
- hypertension: whether the patient has hypertension or not.
- cardiovascular: whether the patient has heart or blood vessels related disease.
- renal chronic: whether the patient has chronic renal disease or not.
- other disease: whether the patient has other disease or not.
- obesity: whether the patient is obese or not.
- tobacco: whether the patient is a tobacco user.
- usmr: Indicates whether the patient treated medical units of the first, second or third level.
- medical unit: type of institution of the National Health System that provided the care.
- intubed: whether the patient was connected to the ventilator.
- icu: Indicates whether the patient had been admitted to an Intensive Care Unit.
- date died: If the patient died indicate the date of death, and 9999-99-99 otherwise.
- fallecidos: 1 for yes and 2 for no.

In [ ]:
```
#dropeamos lo que no me sirve como final clasification

df_covid.drop(['USMER', 'MEDICAL_UNIT', 'CLASIFFICATION_FINAL', 'DATE_DIED'], axis=1, inplace=True)
```

In [ ]:
```
# reemplazo los valores 2 por 0 en todo el dataset

df_covid.replace(2, 0, inplace=True)

df_covid.head()
```

Out[ ]:

	SEX	PATIENT_TYPE	PNEUMONIA	AGE	DIABETES	COPD	ASTHMA	INMSUPR	HIPERTENSION	OTHER_DISEASE	CARD
0	1	1	1.0	65.0	0.0	0.0	0.0	0.0	1.0	0.0	
1	0	1	1.0	72.0	0.0	0.0	0.0	0.0	1.0	0.0	
2	0	0	0.0	55.0	1.0	0.0	0.0	0.0	0.0	0.0	
3	1	1	0.0	53.0	0.0	0.0	0.0	0.0	0.0	0.0	
4	0	1	0.0	68.0	1.0	0.0	0.0	0.0	1.0	0.0	

In [ ]:
```
# creamos el modelo de clasificacion

X = df_covid.drop('fallecidos', axis=1)
y = df_covid['fallecidos']
```

In [ ]:
```
# dividimos el dataset en train y test

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

RED NEURONAL

In [ ]:
```
# Normalizar los datos
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Creamos la RED

Creamos una red basica con 3 capaz y entrenamos durante 10 epocas con tamaño de lotes de 32.

In [ ]:
```
# red neuronal entrenar

model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))
```

```
Epoch 1/10
25621/25621 [==============================] - 27s 1ms/step - loss: 0.1343 - accuracy: 0.9331 - val_los
s: 0.1326 - val_accuracy: 0.9337
Epoch 2/10
25621/25621 [==============================] - 27s 1ms/step - loss: 0.1327 - accuracy: 0.9337 - val_los
s: 0.1327 - val_accuracy: 0.9337
Epoch 3/10
25621/25621 [==============================] - 27s 1ms/step - loss: 0.1326 - accuracy: 0.9339 - val_los
s: 0.1332 - val_accuracy: 0.9336
Epoch 4/10
25621/25621 [==============================] - 28s 1ms/step - loss: 0.1324 - accuracy: 0.9338 - val_los
s: 0.1324 - val_accuracy: 0.9338
Epoch 5/10
25621/25621 [==============================] - 24s 946us/step - loss: 0.1324 - accuracy: 0.9339 - val_lo
ss: 0.1329 - val_accuracy: 0.9340
Epoch 6/10
25621/25621 [==============================] - 24s 952us/step - loss: 0.1323 - accuracy: 0.9339 - val_lo
ss: 0.1323 - val_accuracy: 0.9338
Epoch 7/10
25621/25621 [==============================] - 27s 1ms/step - loss: 0.1323 - accuracy: 0.9340 - val_los
s: 0.1323 - val_accuracy: 0.9338
Epoch 8/10
25621/25621 [==============================] - 26s 1ms/step - loss: 0.1322 - accuracy: 0.9339 - val_los
s: 0.1322 - val_accuracy: 0.9339
Epoch 9/10
25621/25621 [==============================] - 26s 1ms/step - loss: 0.1323 - accuracy: 0.9338 - val_los
s: 0.1321 - val_accuracy: 0.9337
Epoch 10/10
25621/25621 [==============================] - 27s 1ms/step - loss: 0.1322 - accuracy: 0.9337 - val_los
s: 0.1321 - val_accuracy: 0.9340
```

```
Out[ ]: <keras.callbacks.History at 0x2a18ad09de0>
```

Evaluamos

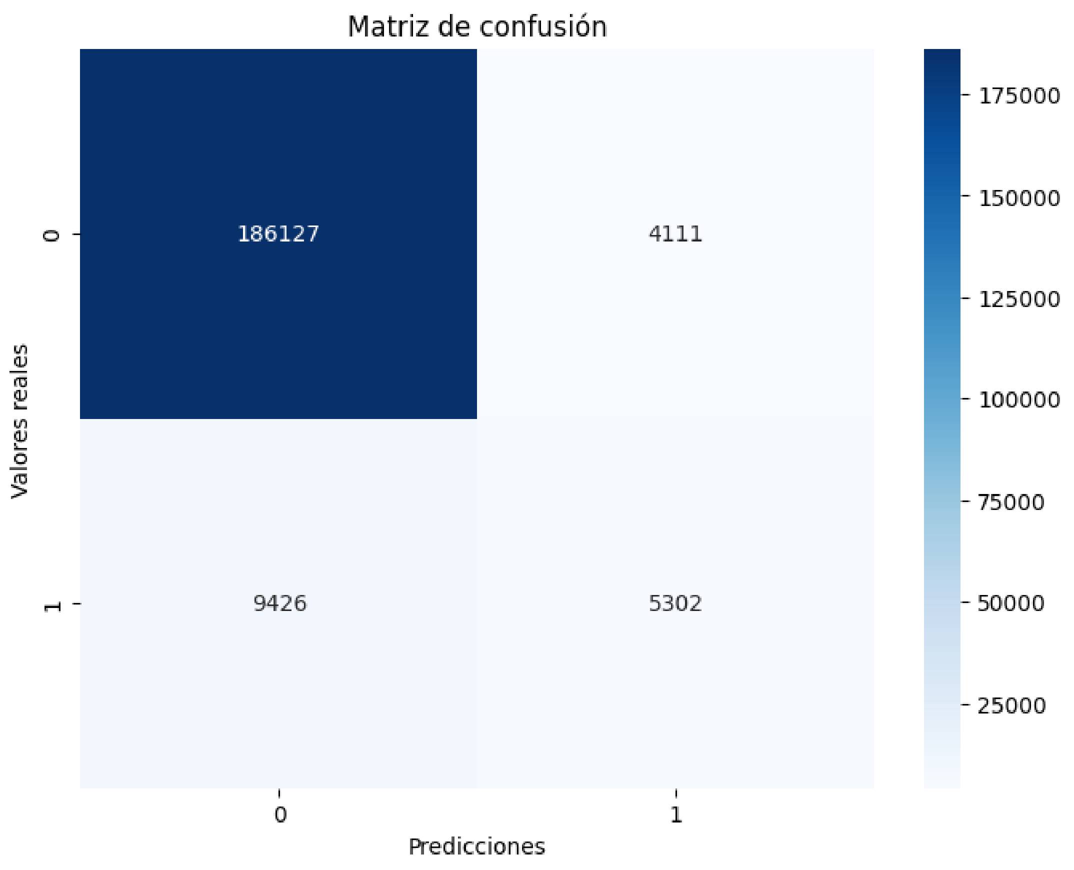
```
In [ ]: #evaluamos el modelo

        loss, accuracy = model.evaluate(X_test, y_test)
        print('Loss:', loss)
        print('Accuracy:', accuracy)
```

```
6406/6406 [==============================] - 4s 649us/step - loss: 0.1321 - accuracy: 0.9340
Loss: 0.1321495920419693
Accuracy: 0.9339548945426941
```

Matriz de confusion

```
In [ ]: # Obtener las predicciones del modelo
        y_pred = model.predict(X_test)
        y_pred = (y_pred > 0.5).astype(int)

        # Calcular la matriz de confusión
        cm = confusion_matrix(y_test, y_pred)

        # Visualizar la matriz de confusión
        plt.figure(figsize=(8, 6))
        sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
        plt.xlabel('Predicciones')
        plt.ylabel('Valores reales')
        plt.title('Matriz de confusión')
        plt.show()
```

```
6406/6406 [==============================] - 4s 589us/step
```

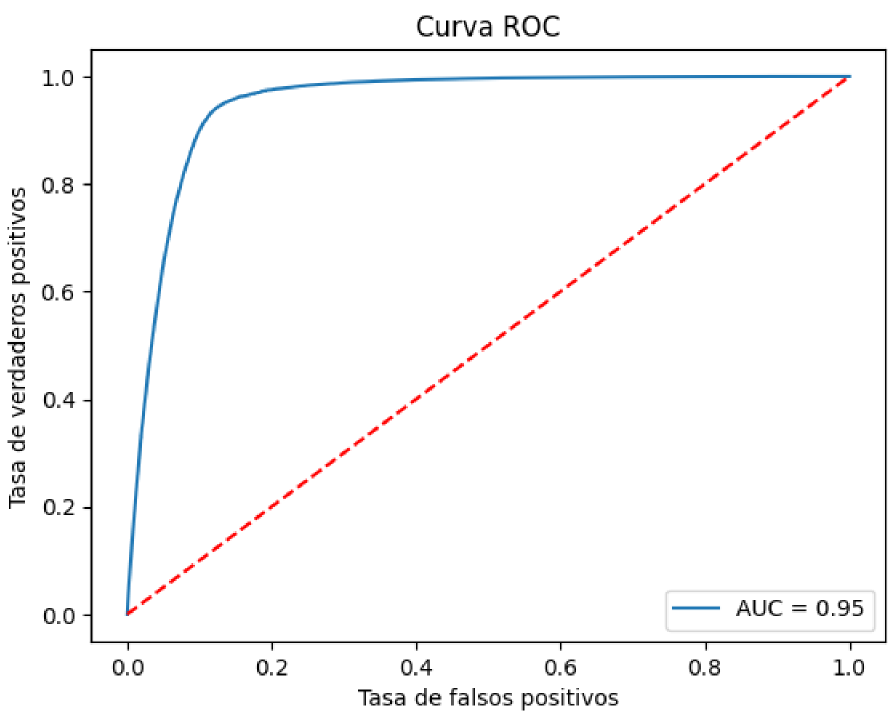


CURVA ROC

Curva ROC (Receiver Operating Characteristic): Es un gráfico que muestra la tasa de verdaderos positivos frente a la tasa de falsos positivos a medida que varías el umbral de clasificación. Puedes utilizar matplotlib o scikit-learn para trazar la curva ROC y calcular el área bajo la curva (AUC-ROC) para evaluar el rendimiento del modelo.

In [ ]:
```
# Calcular las probabilidades de predicción del modelo
y_pred = model.predict(X_test)

# Calcular la tasa de verdaderos positivos y la tasa de falsos positivos
fpr, tpr, thresholds = roc_curve(y_test, y_pred)

# Calcular el área bajo la curva ROC (AUC-ROC)
auc = roc_auc_score(y_test, y_pred)

# Visualizar la curva ROC
plt.plot(fpr, tpr, label=f'AUC = {auc:.2f}')
plt.plot([0, 1], [0, 1], 'r--')
plt.xlabel('Tasa de falsos positivos')
plt.ylabel('Tasa de verdaderos positivos')
plt.title('Curva ROC')
plt.legend()
plt.show()
```

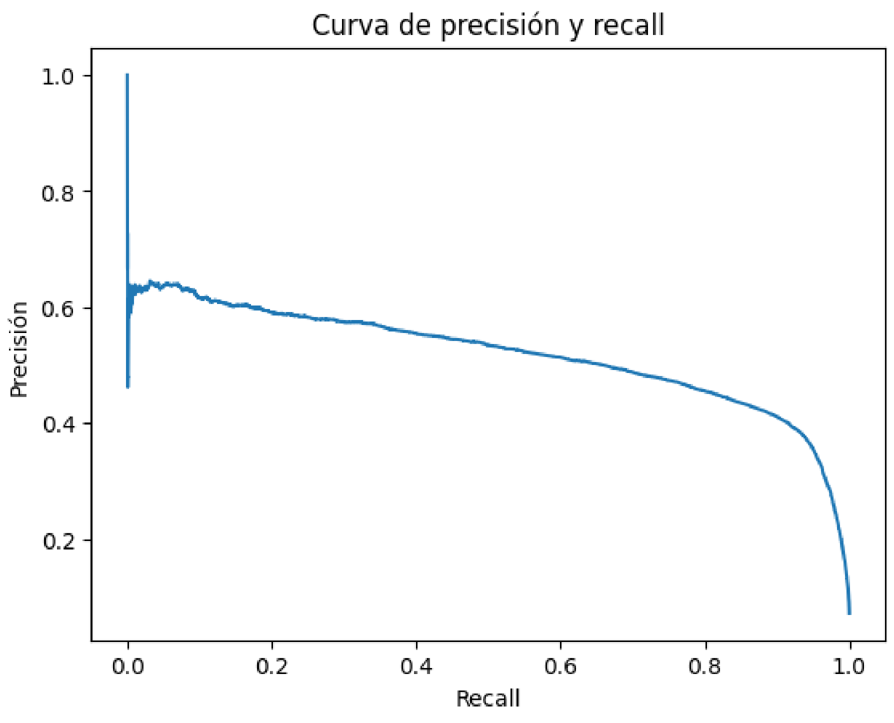
```
6406/6406 [==============================] - 4s 575us/step
```



Curva de Precision y Recall

Curva ROC (Receiver Operating Characteristic): Es un gráfico que muestra la tasa de verdaderos positivos frente a la tasa de falsos positivos a medida que varías el umbral de clasificación. Puedes utilizar matplotlib o scikit-learn para trazar la curva ROC y calcular el área bajo la curva (AUC-ROC) para evaluar el rendimiento del modelo.

```
In [ ]: from sklearn.metrics import precision_recall_curve

# Calcular la precisión y el recall para diferentes umbrales
precision, recall, thresholds = precision_recall_curve(y_test, y_pred)

# Visualizar la curva de precisión y recall
plt.plot(recall, precision)
plt.xlabel('Recall')
plt.ylabel('Precisión')
plt.title('Curva de precisión y recall')
plt.show()
```



random forest

In [ ]:
```
# dividimos el dataset en train y test

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Entrenamos un modelo de random forest con 100 arboles y evaluamos

In [ ]:
```
# entrenamos el modelo

rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
```

Out[ ]:
```
▾        RandomForestClassifier
RandomForestClassifier(random_state=42)
```

Evaluamos

In [ ]:
```
# predecimos

y_pred = rf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy: ', accuracy)
```

```
Accuracy:  0.9307250958695589
```

Vemos la importancia de cada variable

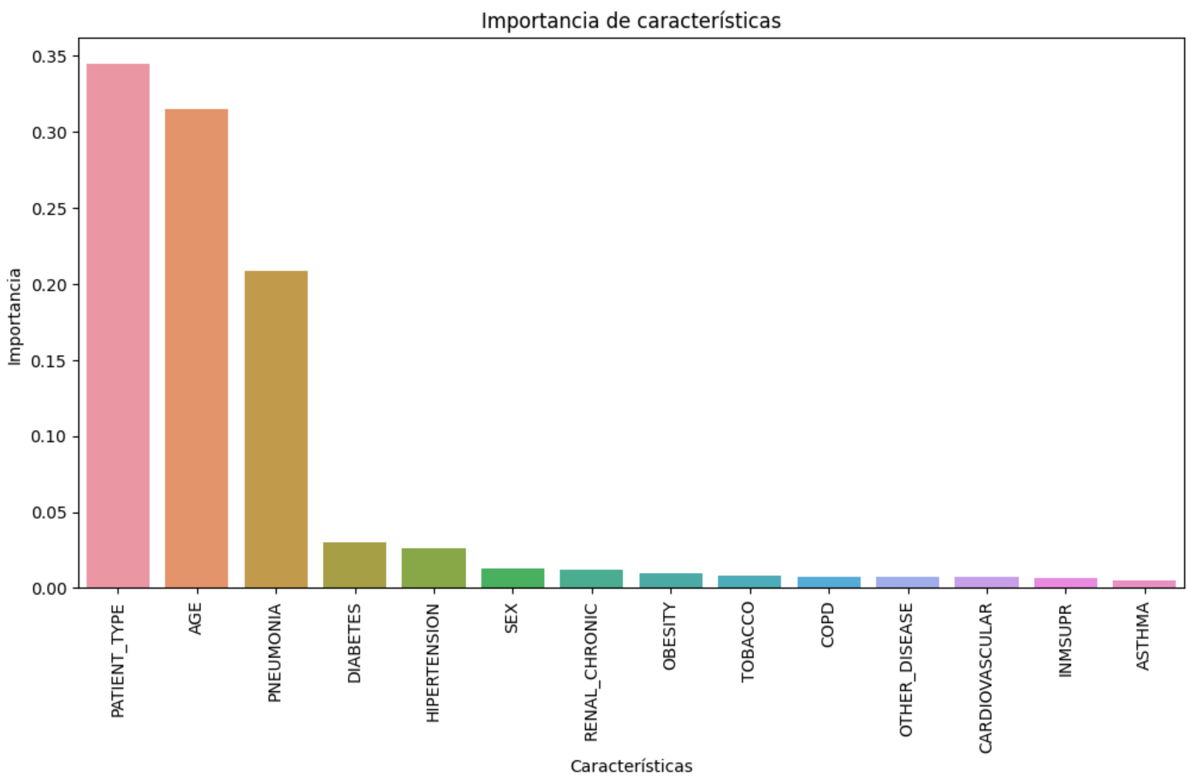
In [ ]:
```
# vemos la importancia de las variables

importances = pd.DataFrame({'feature':X_train.columns,'importance':np.round(rf.feature_importances_,3)})
importances = importances.sort_values('importance',ascending=False).set_index('feature')
print(importances)
```

```
                importance
feature
PATIENT_TYPE         0.344
AGE                  0.315
PNEUMONIA            0.209
DIABETES             0.030
HIPERTENSION         0.026
SEX                  0.013
RENAL_CHRONIC        0.012
OBESITY              0.010
COPD                 0.008
OTHER_DISEASE        0.008
TOBACCO              0.008
INMSUPR              0.007
CARDIOVASCULAR       0.007
ASTHMA               0.005
```

Hacemos un grafico de la importancia de cada variable en el modelo para comprender que variables es mas importante para el modelo y para el resultado buscado.

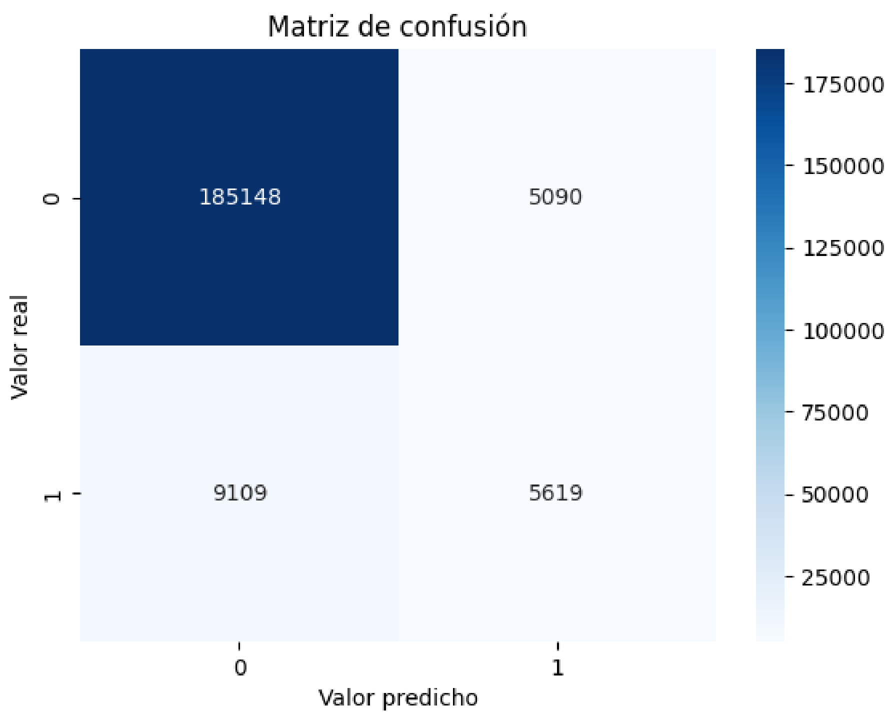
In [ ]:
```
# Obtener los valores de importancia de características y sus nombres

feature_importances = rf.feature_importances_
feature_names = X_train.columns

# Crear un dataframe con los valores de importancia de características y sus nombres

feature_importances_df = pd.DataFrame({'feature_importances': feature_importances,
                                        'feature_names': feature_names})

# Ordenar los valores de importancia de características de mayor a menor

feature_importances_df.sort_values('feature_importances', ascending=False, inplace=True)

# Crear un gráfico de barras con los valores de importancia de características

plt.figure(figsize=(12, 6))
```

```
sns.barplot(x=feature_importances_df.feature_names, y=feature_importances_df.feature_importances)

plt.title('Importancia de características')

plt.xlabel('Características')

plt.ylabel('Importancia')

plt.xticks(rotation=90)

plt.show()
```



Matriz de confusion

```
In [ ]: #creamos matriz de confusion

        cm = confusion_matrix(y_test, y_pred)

        #creamos el heatmap

        sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

        plt.title('Matriz de confusión')

        plt.xlabel('Valor predicho')

        plt.ylabel('Valor real')

        plt.show()
```



Conclusiones:

Como podes ver, el modelo de red neuronal tiene un accuracy de 0.93 y el modelo de random forest tiene un accuracy de 0.93. Ambos metodos nos dan un resultado similar. Lo que nos dice que el modelo de red neuronal no es mejor que el modelo de random forest. Pero tambien que 0.93 es un resultado posiblemente mejorable.

- Red Neuronal

6406/6406 [==============================] - 4s 649us/step loss: 0.1321 - accuracy: 0.9340

```
Loss: 0.1321495920419693

Accuracy: 0.9339548945426941
```

- Random Forest

  Accuracy: 0.9307250958695589